mitp

Robert C. Martin

Mit Vorwort von Stacia Heimgartner Viscardi

Deutsche Ausgabe

Robert C. Martin Series

Clean Craftsmanship

Best Practices, Standards und Ethik für die Softwareentwicklung

Stimmen zum Buch

»Bobs *Clean Craftsmanship* hat großartige Arbeit dabei geleistet, den Zweck agiler technischer Praktiken zu erklären, zusammen mit einer tiefen historischen Basis, die aufzeigt, wie sie entstanden sind, sowie einer Positionierung, wieso er davon ausgeht, dass sie immer von Bedeutung sein werden. Seine Beteiligung an der Geschichte und der Entstehung von Agilität, sein umfassendes Verständnis der Praktiken und ihres Sinns spiegeln sich im gesamten Manuskript deutlich wider.«

– Tim Ottinger, bekannter Agile Coach und Autor.

»Bobs Schreibstil ist ausgezeichnet. Er ist leicht zu lesen, und die Konzepte werden so detailliert erklärt, dass auch ein neuer Programmierer sie verstehen kann. Bob hat sogar einige lustige Anekdoten parat, die einen beim Lesen angenehm aus dem Konzept bringen. Der wahre Wert des Buchs liegt in dem Schrei nach Veränderung, nach etwas Besserem ... dem Schrei nach professionellen Programmierern ... der Erkenntnis, dass Software überall ist. Außerdem glaube ich, dass die Historie, die Bob bereitstellt, von großem Wert ist. Mir gefällt, dass er keine Zeit mit Schuldzuweisungen verschwendet, wer dafür verantwortlich ist, dass wir dahin gekommen sind, wo wir jetzt sind. Bob ruft die Menschen zum Handeln auf und fordert sie auf, Verantwortung zu übernehmen, indem sie ihre Standards und ihr Niveau an Professionalität erhöhen, auch wenn das manchmal bedeutet, sich zu wehren.«

– Heather Kanser

»Als Softwareentwickler müssen wir ständig wichtige Probleme für unsere Arbeitgeber, Kunden, Kollegen und unser zukünftiges Ich lösen. Eine App zum Laufen zu bringen, auch wenn es schwierig ist, ist nicht genug, es macht Sie nicht zum Craftsman. Wenn eine App funktioniert, haben Sie den Eignungstest bestanden. Sie haben vielleicht das Potenzial, ein Craftsman zu werden, aber es gibt noch mehr, was Sie beherrschen müssen. Auf diesen Seiten beschreibt Bob klar und deutlich die Techniken und Verantwortlichkeiten, um über den App-Eignungstest hinauszugehen und zeigt den Weg, wie Sie ein ernsthafter Software Craftsman werden.«

– James Grenning, Autor von »Test-Driven Development for Embedded C« und Mitverfasser des Agilen Manifests

»Bob ist einer der wenigen berühmten Entwickler, mit denen ich gerne an einem technischen Projekt arbeiten würde. Nicht, weil er ein guter Entwickler, berühmt oder ein guter Kommunikator ist, sondern weil er mir hilft, ein besserer Entwickler und ein besseres Teammitglied zu sein. Er hat jeden wichtigen Entwicklungstrend Jahre vor allen anderen erkannt und war in der Lage, dessen Tragweite zu erklären, was mich zum Lernen ermutigte. Als ich anfing, fehlte in diesem Bereich – abgesehen davon, ehrlich und ein guter Mensch zu sein – jede Vorstellung von Handwerkskunst und Ethik. Jetzt scheint es das Wichtigste zu sein, was professionelle Entwickler lernen können, sogar noch vor dem Programmieren selbst. Ich freue mich, dass Bob wieder den Weg vorgibt. Ich kann es kaum erwarten, seine Perspektive zu hören und sie in meine eigene Praxis einzubringen.«

– Daniel Markham, Principal, Bedford Technology Group, Inc.

In Gedenken an Mike Beedle

Robert C. Martin

Clean Craftsmanship

Best Practices, Standards und Ethik für die Softwareentwicklung

Übersetzung aus dem Englischen
von Uwe M. Schirmer

mitp

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN 978-3-7475-0417-8
1. Auflage 2022

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

Lektorat: Janina Bahlmann
Sprachkorrektorat: Christine Hoffmeister
Covergestaltung: Christian Kalkert
Bildnachweis: Richie_K / stock.adobe.com
Satz: III-satz, www.drei-satz.de

Inhaltsverzeichnis

Vorwort 13

Vorwort des Übersetzers der deutschen Ausgabe 17

Einleitung 19
Über den Begriff »Craftsmanship« 19
Auf dem einzig wahren Weg 19
Einführung in das Buch 20

Danksagungen 25

Über den Autor 27

1 **Craftsmanship** 29

Teil I Die Praktiken 37

2 **Testgetriebene Entwicklung** 45
2.1 Überblick 46
2.1.1 Software 48
2.1.2 Die drei Gesetze des TDD 49
2.1.3 Das vierte Gesetz 57
2.2 Die Grundlagen 59
2.2.1 Einfache Beispiele 59
2.2.2 Stack 60
2.2.3 Primfaktoren 75
2.2.4 Bowling 83
2.3 Fazit 100

3 **Fortgeschrittenes TDD** 101
3.1 Sort 1 101
3.2 Sort 2 106
3.3 Steckenbleiben 114
3.4 Erstellen, Ausführen, Sicherstellen 121
3.4.1 BDD einführen 122

3.4.2 Endliche Automaten . . . 122
3.4.3 Und wieder BDD . . . 124
3.5 Test-Doubles . . . 124
3.5.1 Dummy . . . 127
3.5.2 Stub . . . 130
3.5.3 Spy . . . 133
3.5.4 Mock . . . 135
3.5.5 Fake . . . 138
3.5.6 Das TDD-Unsicherheitsprinzip . . . 140
3.5.7 London vs. Chicago . . . 152
3.6 Architektur . . . 155
3.7 Fazit . . . 157

4 Testdesign . . . 159
4.1 Datenbanken testen . . . 160
4.2 Benutzeroberflächen testen . . . 161
4.2.1 Eingaben in der GUI . . . 163
4.3 Test Pattern . . . 164
4.3.1 Testspezifische Unterklasse . . . 165
4.3.2 Self-Shunt . . . 166
4.3.3 Humble Object . . . 166
4.4 Testdesign . . . 169
4.4.1 Das Problem der fragilen Tests . . . 169
4.4.2 Die Eins-zu-eins-Kopplung . . . 170
4.4.3 Durchbrechen der Kopplung . . . 171
4.4.4 Die Videothek . . . 173
4.4.5 Speziell versus allgemein . . . 190
4.5 Transformationsprioritätsgrundsatz . . . 191
4.5.1 {} → Nil . . . 193
4.5.2 Nil → Konstante . . . 193
4.5.3 Konstante → Variable . . . 194
4.5.4 Ohne Bedingung → Verzweigung . . . 195
4.5.5 Wert → Liste . . . 195
4.5.6 Anweisung → Rekursion . . . 196
4.5.7 Verzweigung → Iteration . . . 196
4.5.8 Wert → Veränderter Wert . . . 197
4.5.9 Beispiel: Fibonacci . . . 197
4.5.10 Die Prämisse der Priorität der Transformation . . . 201
4.6 Fazit . . . 202

5 Refactoring . . . 203
5.1 Was ist Refactoring? . . . 204
5.2 Das Basis-Toolkit . . . 205
5.2.1 Umbenennen . . . 205
5.2.2 Methode extrahieren . . . 206
5.2.3 Variable extrahieren . . . 207
5.2.4 Feld extrahieren . . . 209
5.2.5 Rubiks Würfel . . . 221
5.3 Die Praktiken . . . 221
5.3.1 Tests . . . 221
5.3.2 Schnelle Tests . . . 222
5.3.3 Aufbrechen tiefreichender Eins-zu-eins-Kopplungen . . . 222
5.3.4 Kontinuierliches Refactoring . . . 222
5.3.5 Gnadenloses Refactoring . . . 223
5.3.6 Lassen Sie die Tests bestehen! . . . 223
5.3.7 Lassen Sie sich einen Ausweg offen . . . 224
5.4 Fazit . . . 224

6 Einfaches Design . . . 225
6.1 YAGNI . . . 227
6.2 Abgedeckt durch Tests . . . 229
6.2.1 Abdeckung . . . 230
6.2.2 Ein asymptotisches Ziel . . . 231
6.2.3 Design? . . . 232
6.2.4 Aber da ist mehr . . . 232
6.3 Aussagekraft maximieren . . . 233
6.3.1 Die zugrunde liegende Abstraktion . . . 234
6.3.2 Tests: Die zweite Hälfte des Problems . . . 235
6.4 Duplikate minimieren . . . 236
6.5 Zufällige Duplizierung . . . 237
6.6 Größe minimieren . . . 238
6.7 Einfaches Design . . . 238

7 Kollaborative Programmierung . . . 239

8 Akzeptanztests . . . 243
8.1 Die Praktiken . . . 245
8.2 Der kontinuierliche Build . . . 246

Teil II Die Standards 247

9 Produktivität 249
9.1 Wir werden nie Sch**** ausliefern 249
9.2 Leichte Anpassbarkeit 251
9.3 Wir werden immer bereit sein 252
9.4 Stabile Produktivität 254

10 Qualität 255
10.1 Kontinuierliche Verbesserung 255
10.2 Furchtlose Kompetenz 256
10.3 Extreme Qualität 257
10.4 Wir werfen die QS nicht über Bord 258
10.4.1 Die QS-Krankheit 259
10.5 Die QS wird nichts finden 259
10.6 Testautomatisierung 260
10.7 Automatisiertes Testen und Benutzeroberflächen 260
10.8 Testen der Benutzeroberfläche 261

11 Mut 263
11.1 Wir stehen füreinander ein 263
11.2 Ehrliche Schätzungen 264
11.3 Sie müssen NEIN sagen 266
11.4 Ständiges aggressives Lernen 267
11.5 Mentoring 268

Teil III Die Ethik 269

12 Schaden 283
12.1 Erstens, keinen Schaden anrichten 284
12.1.1 Gesellschaftlicher Schaden 285
12.1.2 Funktionsbeeinträchtigung 286
12.1.3 Keine Schädigung der Struktur 288
12.1.4 Soft 290
12.1.5 Tests 291
12.2 Beste Arbeit 292
12.2.1 Es richtig machen 293
12.2.2 Was ist eine gute Struktur? 294

12.2.3 Eisenhower-Matrix 295
12.2.4 Programmierer sind Stakeholder 297
12.2.5 Ihr Bestes 298
12.3 Reproduzierbarer Beweis 300
12.3.1 Dijkstra 300
12.3.2 Beweis der Korrektheit 301
12.3.3 Strukturierte Programmierung 303
12.3.4 Funktionale Dekomposition 305
12.3.5 Testgetriebene Entwicklung 306

13 Integrität 309
13.1 Kleine Zyklen 309
13.1.1 Die Geschichte der Versionsverwaltung 310
13.1.2 Git 314
13.1.3 Kurze Zyklen 315
13.1.4 Kontinuierliche Integration 316
13.1.5 Branches versus Toggles 317
13.1.6 Kontinuierliches Deployment 319
13.1.7 Kontinuierlicher Build 320
13.2 Unerbittliche Verbesserung 321
13.2.1 Testabdeckung 321
13.2.2 Mutationstests 322
13.2.3 Semantische Stabilität 322
13.2.4 Aufräumen 323
13.2.5 Kreationen 323
13.3 Hohe Produktivität beibehalten 324
13.3.1 Viskosität 324
13.3.2 Umgang mit Ablenkungen 327
13.3.3 Zeitmanagement 329

14 Teamarbeit 331
14.1 Arbeit im Team 331
14.1.1 Offenes/virtuelles Büro 332
14.2 Ehrliche und faire Schätzungen 333
14.2.1 Lügen 334
14.2.2 Ehrlichkeit, Genauigkeit, Präzision 334
14.2.3 Genauigkeit 339
14.2.4 Präzision 340

14.2.5 Aggregation 342
14.2.6 Ehrlichkeit 343
14.3 Respekt 345
14.4 Niemals aufhören zu lernen 345

Stichwortverzeichnis 347

Vorwort

Ich erinnere mich, dass ich Uncle Bob im Frühjahr 2003 traf, kurz nachdem Scrum in unserem Unternehmen und in unseren Technologieteams eingeführt worden war. Als skeptischer, junger Scrum Master erinnere ich mich daran, dass ich Bob zuhörte, als er uns in TDD und einem kleinen Tool namens FitNesse unterrichtete, und ich erinnere mich, dass ich mir dachte: »Warum sollten wir jemals Testfälle schreiben, die zuerst fehlschlagen? Kommt das Testen nicht erst *nach* dem Coding?« Ich habe mich oft am Kopf gekratzt, wie viele meiner Teammitglieder, und doch erinnere ich mich bis heute deutlich an Bobs spürbare Begeisterung für das Code-Handwerk, als wäre es erst gestern gewesen. Ich erinnere mich an seine Direktheit, als er sich eines Tags unseren Bug-Backlog ansah und uns fragte, warum in aller Welt wir so schlechte Entscheidungen über Softwaresysteme treffen würden, die uns eigentlich nicht gehörten – »Diese Systeme sind *Firmeneigentum*, nicht euer *persönliches Eigentum*.« Seine Leidenschaft machte uns neugierig, und anderthalb Jahre später hatten wir durch Refactoring eine automatisierte Testabdeckung von etwa 80 % und eine saubere Codebasis, die uns Anpassungen wesentlich erleichterte und zu zufriedeneren Kunden – und zufriedeneren Teams – führte. Danach kamen wir blitzschnell voran und nutzten unsere »*Definition of done*« wie eine Rüstung, um uns vor den immer lauernden Codekobolden zu schützen; wir hatten im Wesentlichen gelernt, wie wir uns vor uns selbst schützen konnten. Mit der Zeit entwickelten wir eine große Sympathie für Uncle Bob, der sich für uns immer mehr wie ein richtiger Onkel anfühlte – ein warmherziger, entschlossener und mutiger Mann, der uns mit der Zeit half, zu lernen, für uns selbst einzustehen und das Richtige zu tun. Während die Onkel mancher Kinder ihnen beibrachten, wie man Fahrrad fährt oder angelt, lehrte uns unser Uncle Bob, unsere Integrität nicht aufs Spiel zu setzen – und bis heute ist die Fähigkeit und der Wunsch, jeder Situation mit Mut und Neugierde zu begegnen, die beste Lektion meiner Karriere.

Ich nahm Bobs frühe Lektionen mit auf meine Reise, als ich mich als agiler Coach in die Welt hinauswagte und habe schnell selbst beobachtet, dass die besten Produktentwicklungsteams herausfanden, wie sie ihre eigenen Best Practices für ihre einzigartigen Kontexte, für ihre speziellen Kunden, in ihren jeweiligen Branchen zusammenstellen konnten. Ich erinnerte mich an Bobs Lektionen, als ich feststellte, dass die besten Entwicklungswerkzeuge der Welt nur so gut sind wie ihre menschlichen Bediener – die Teams, die die besten *Anwendungen* dieser Werkzeuge in ihren eigenen Domänen fanden. Ich beobachtete, dass Teams einen hohen Pro-

zentsatz an Unit-Test-Abdeckung erreichen können, um den Punkt abzuhaken und die Metrik zu erfüllen, nur um dann feststellen, dass ein großer Prozentsatz dieser Tests fehlerhaft war – die Metrik wurde erfüllt, aber es wurde kein Wert erbracht. Die besten Teams brauchten sich nicht wirklich um Metriken zu kümmern; sie hatten ein Ziel, Disziplin, Stolz und Verantwortung – und die Metriken sprachen in jedem Fall für sich selbst. *Clean Craftsmanship* verwebt all diese Lektionen und Prinzipien mit praktischen Codebeispielen und Erfahrungen, um den Unterschied zu verdeutlichen, zwischen etwas zu schreiben, um eine Frist einzuhalten, und dem tatsächlichen Aufbau von etwas Nachhaltigem für die Zukunft.

Clean Craftsmanship erinnert uns daran, uns niemals mit weniger zufrieden zu geben und mit *furchtloser Kompetenz* über diese Erde zu wandeln. Dieses Buch wird Sie wie ein alter Freund daran erinnern, worauf es ankommt, was funktioniert, was nicht funktioniert, was Risiken schafft und was sie verringert. Diese Lektionen sind zeitlos. Vielleicht stellen Sie fest, dass Sie einige der darin enthaltenen Techniken bereits praktizieren, und ich wette, Sie werden etwas Neues finden oder zumindest etwas, das Sie fallengelassen haben, weil Sie irgendwann vor Terminen oder anderem Druck in Ihrer Karriere kapituliert haben. Wenn Sie neu in der Welt der Entwicklung sind – sei es im Management oder in der Technik – werden Sie vom Besten lernen, und selbst die Geübtesten und Kampfesmüden werden Wege finden, sich zu verbessern. Vielleicht hilft Ihnen dieses Buch, Ihre Leidenschaft wiederzuentdecken, Ihren Wunsch zu erneuern, Ihr Handwerk zu verbessern oder Ihre Energie erneut der Suche nach Perfektion zu widmen, ungeachtet der Hindernisse, die sich Ihnen in den Weg stellen.

Softwareentwickler regieren die Welt, und Uncle Bob ist wieder da, um uns an die professionellen Praktiken derjenigen zu erinnern, die so viel Macht haben. Er macht dort weiter, wo er mit *Clean Code* aufgehört hat. Da Softwareentwickler buchstäblich die Regeln der Menschheit schreiben, erinnert uns Uncle Bob daran, dass wir einen strengen ethischen Kodex einhalten müssen, eine Verantwortung haben, zu wissen, was der Code macht, wie die Menschen ihn verwenden und wo er bricht. Softwarefehler kosten Menschen ihren Lebensunterhalt – und ihr Leben. Software beeinflusst die Art und Weise, wie wir denken, die Entscheidungen, die wir treffen, und durch künstliche Intelligenz und prognostische Analytik beeinflusst sie das Sozial- und Herdenverhalten. Daher müssen wir verantwortungsbewusst sein und mit großer Sorgfalt und Empathie handeln – die Gesundheit und das Wohlbefinden der Menschen hängen davon ab. Uncle Bob hilft uns, dieser Verantwortung gerecht zu werden, und er hilft uns, *die Fachleute zu werden, die unsere Gesellschaft von uns erwartet und verlangt*.

Da sich das Agile Manifest zum Zeitpunkt der Verfassung dieses Vorworts seinem zwanzigsten Geburtstag nähert, ist dieses Buch eine perfekte Gelegenheit, zu den Grundlagen zurückzukehren: zum richtigen Zeitpunkt eine bescheidene Erinnerung an die ständig zunehmende Komplexität unserer programmatischen Welt

und daran, dass wir es dem Vermächtnis der Menschheit – und uns selbst – schuldig sind, ethische Entwicklung zu betreiben. Nehmen Sie sich die Zeit, *Clean Craftsmanship* zu lesen. Verinnerlichen Sie sich die Prinzipien. Üben Sie sie. Verbessern Sie sie. Leiten Sie andere an. Behalten Sie dieses Buch in Ihrem Bücherregal. Lassen Sie dieses Buch *Ihr* alter Freund sein – *Ihr* Uncle Bob, Ihr Wegweiser – wenn Sie sich mit Neugier und Mut auf den Weg durch diese Welt machen.

– Stacia Heimgartner Viscardi, CST & Agile Mentor

Vorwort des Übersetzers der deutschen Ausgabe

Ich hatte leider noch nicht die Gelegenheit, Robert C. Martin persönlich kennenzulernen. Trotzdem kenne ich, wie viele Softwareentwickler und Softwarearchitekten in meinem Umfeld, die meisten seiner Werke. Deswegen war ich auch hocherfreut, als ich gefragt wurde, ob ich sein neuestes Buch übersetzen könne.

Ich muss zugeben, dass ich Martins Bücher bisher auf Englisch gelesen hatte. Ich schreibe Bücher auf Deutsch und Englisch und habe auch schon Bücher von Freunden übersetzt. Clean Craftsmanship zu übersetzen, war aber ein Abenteuer für sich. Uncle Bob verwendet eine sehr lebendige Sprache und spielt mit Wörtern und Begriffen. Beim Übersetzen eines englischen Begriffs muss man sich oft für eine Facette, eine Bedeutung entscheiden. Das wäre an einigen Stellen den Wortspielereien von Robert C. Martin nicht gerecht geworden. Deswegen haben wir im Buch einige Begriffe nicht übersetzt, z.B. das titelgebende Craftsmanship oder einige Begriffe aus der IT, die auch den meisten deutschen »Softwerkern« geläufig sein sollten. Diese Begriffe zu übersetzen, hätte mehr Verwirrung gestiftet, als Klarheit gebracht. Ich bin mir bewusst, dass dies eine subjektive Entscheidung ist und einige Leser anderer Ansicht sein werden. Seien Sie aber versichert, dass wir über jeden der hier verwendeten Fachbegriffe lange diskutiert haben, ob wir ihn übersetzen oder belassen, und dies auch mit früheren Übersetzungen von Robert C. Martins Büchern abgeglichen haben.

Ich bin selbst seit mehr als 20 Jahren in der Softwareentwicklung tätig und musste bei einigen seiner Beispiele aus frühen Jahren schmunzeln, weil ich sie teilweise noch selbst miterleben durfte. Auch wenn ich gottlob nie Programme auf Lochkarten erstellen musste, habe ich doch zumindest in meinen frühen Jahren als Entwickler einmal Lochkarten in der Hand gehabt. Und die frühen Versionsverwaltungssysteme habe ich selbst noch als Entwickler verwendet. Auch wenn diese Beispiele aus einer anderen Zeit zu stammen scheinen, bringen sie einem doch die Entwicklung unseres Handwerks näher und tragen dazu bei, dass wir besser verstehen, warum manche Dinge heute so sind, wie sie sind.

Ich bin selbst ein großer Fan von testgetriebener Entwicklung, auch wenn ich meine Leidenschaft in echten Projekten oft nicht so ausleben kann, wie ich es gerne tun würde. Dazu sind Projekte oft zu sehr ein Zusammenspiel zahlreicher Individuen, und die testgetriebene Entwicklung ist nicht so weitverbreitet, wie sie

es sein sollte. Robert C. Martin liefert in diesem Buch ein paar sehr gelungene Beispiele, wie Testgetriebene Entwicklung in der Praxis funktioniert, und er liefert auch sehr gute Argumente, warum man testgetrieben entwickeln sollte. Argumente für Entwickler und Manager. Ich muss zugeben, dass es mir oft nicht leicht gefallen ist, die Testgetriebene Entwicklung in Projekten zu verteidigen. Nach dem Übersetzen und Lesen dieses Buchs bin ich mir aber sicher, dass es mir leichter fallen wird, dies in Zukunft zu tun.

Ich wünsche Ihnen viel Spaß beim Lesen und Lernen.

– Uwe M. Schirmer
Software Architekt, Autor und Übersetzer

Einleitung

Bevor wir beginnen, müssen wir uns mit zwei Fragen befassen, um sicherzustellen, dass Sie, verehrte Leserinnen und Leser, den Rahmen verstehen, in dem dieses Buch präsentiert wird.

Über den Begriff »Craftsmanship«

Der Beginn des 21. Jahrhunderts war von einer Kontroverse über die Sprache geprägt. Auch wir in der Softwarebranche hatten unseren Anteil an dieser Kontroverse. Ein Begriff, der oft als nicht inklusiv bezeichnet wird, ist *Craftsman* (dt. Handwerker).

Ich habe lange über diese Frage nachgedacht und mit vielen Menschen mit unterschiedlichen Meinungen gesprochen, und ich bin zu dem Schluss gekommen, dass es keinen besseren Begriff gibt, der im Zusammenhang mit diesem Buch verwendet werden kann

Es wurden Alternativen zu Craftsman in Betracht gezogen, darunter Craftsperson, Craftsfolk und Crafter. Aber keiner dieser Begriffe hat das historische Gewicht des *Craftsman*. Und diese historische Bedeutung ist wichtig für die Botschaft dieses Buchs.

Mit dem Begriff *Craftsman* wird eine Person assoziiert, die in einer bestimmten Tätigkeit sehr geschickt und versiert ist – jemand, der sich mit seinen Werkzeugen und seinem Handwerk auskennt, der stolz auf seine Arbeit ist und von dem man erwarten kann, dass er seinen Beruf mit Würde und Professionalität ausübt.

Es mag sein, dass einige von Ihnen mit meiner Entscheidung nicht einverstanden sind. Ich verstehe, warum das so sein könnte. Ich hoffe nur, dass Sie dies nicht als Versuch interpretieren, in irgendeiner Weise exklusiv zu sein – denn das ist keineswegs meine Absicht.

Auf dem einzig wahren Weg

Wenn Sie *Clean Craftsmanship: Best Practices, Standards und Ethik für die Softwareentwicklung lesen*, bekommen Sie vielleicht das Gefühl, dass dies *der einzig wahre Weg zur Craftsmanship* ist. *Für mich* mag er das sein, aber nicht unbedingt für Sie.

Ich biete Ihnen dieses Buch als ein Beispiel für *meinen Weg* an. Sie werden natürlich Ihren eigenen Weg finden müssen.

Werden wir irgendwann den einen *einzigen wahren Weg* brauchen? Ich weiß es nicht. Aber vielleicht. Wie Sie auf diesen Seiten lesen werden, wächst der Druck für eine strenge Definition des Softwareberufs. Je nach der Kritikalität der zu erstellenden Software können wir vielleicht mit mehreren verschiedenen Wegen auskommen. Aber wie Sie im Folgenden lesen werden, ist es vielleicht gar nicht so einfach, kritische von unkritischer Software zu unterscheiden.

Über eines bin ich mir aber sicher. Die Zeiten der »Richter«[1] sind vorbei. Es reicht nicht mehr aus, dass jeder Programmierer das tut, was er in seinen Augen für richtig hält. Es *werden* Praktiken, Standards und Ethik kommen. Die Entscheidung, vor der wir heute stehen, ist, ob wir Programmierer sie für uns selbst definieren oder ob wir sie uns von denen aufzwingen lassen wollen, die uns nicht kennen.

Einführung in das Buch

Dieses Buch ist für Programmierer und für Manager von Programmierern geschrieben. Aber in einem anderen Sinne ist dieses Buch für die gesamte menschliche Gesellschaft geschrieben. Denn wir, die Programmierer, haben uns ungewollt am Dreh- und Angelpunkt dieser Gesellschaft wiedergefunden.

Für Sie selbst

Wenn Sie ein Programmierer mit mehreren Jahren Erfahrung sind, kennen Sie wahrscheinlich die Genugtuung, wenn Sie ein System einrichten und zum Laufen bringen. Es erfüllt Sie mit einem gewissen Stolz, an einer solchen Leistung beteiligt gewesen zu sein. Sie sind stolz darauf, dass Sie das System auf den Weg gebracht haben.

Aber sind Sie auch stolz darauf, *wie Sie* das System auf den Weg gebracht haben? Ist es der Stolz darauf, fertig geworden zu sein? Oder ist es der Stolz auf Ihre Arbeit? Sind Sie stolz darauf, dass das System deployt wurde? Oder sind Sie stolz darauf, wie Sie das System gebaut haben?

Wenn Sie nach einem anstrengenden Tag, an dem Sie Code geschrieben haben, nach Hause gehen, schauen Sie sich dann im Spiegel an und sagen: »Heute habe ich gute Arbeit geleistet«? Oder müssen Sie erst duschen gehen?

Zu viele von uns fühlen sich am Ende des Tags schmutzig. Zu viele von uns fühlen sich dazu gezwungen, minderwertige Arbeit zu leisten. Zu viele von uns haben

1 eine Anspielung auf das alttestamentarische Buch der Richter

das Gefühl, dass niedrige Qualität erwartet wird und für hohe Geschwindigkeit notwendig ist. Zu viele von uns denken, dass Produktivität und Qualität in einem umgekehrten Verhältnis zueinanderstehen.

In diesem Buch versuche ich, diese Denkweise zu durchbrechen. Dies ist ein Buch darüber, *gut zu arbeiten*. Es ist ein Buch darüber, seinen Job gut zu machen. Es ist ein Buch, das die Praktiken beschreibt, die jeder Programmierer kennen sollte, um schnell zu arbeiten, produktiv zu sein und stolz auf das, was er jeden Tag schreibt.

Für die Gesellschaft

Das 21. Jahrhundert markiert das erste Mal in der Geschichte der Menschheit, dass unsere Gesellschaft für ihr Überleben von einer Technologie abhängig geworden ist, die praktisch keinen Anschein von Disziplin oder Kontrolle mehr hat. Software ist in jede Facette des modernen Lebens eingedrungen, vom morgendlichen Kaffeekochen bis zur abendlichen Unterhaltung, vom Wäschewaschen bis zum Autofahren, vom Verbinden über ein weltumspannendes Netz bis zur Aufspaltung in gesellschaftliche und politische Lager. Es gibt buchstäblich keinen Aspekt des Lebens in der modernen Welt, der nicht von Software beherrscht wird. Und doch sind diejenigen von uns, die diese Software entwickeln, kaum mehr als ein zusammengewürfelter Haufen von Tüftlern, die nur wenig Ahnung davon haben, was sie da tun.

Hätten wir Programmierer besser verstanden, was wir tun, wären dann die Ergebnisse der Caucus-Wahl in Iowa 2020 zum versprochenen Zeitpunkt fertig gewesen? Wären bei den beiden Abstürzen der 737 Max 346 Menschen ums Leben gekommen? Hätte die Knight Capital Group in 45 Minuten 460 Millionen Dollar verloren? Hätten 89 Menschen bei den Unfällen mit unbeabsichtigter Beschleunigung bei Toyota ihr Leben verloren?

Alle fünf Jahre verdoppelt sich die Zahl der Programmierer auf der Welt. Diesen Programmierern wird nur sehr wenig über ihr Handwerk beigebracht. Man zeigt ihnen die Werkzeuge, gibt ihnen ein paar Spielzeugprojekte, die sie entwickeln sollen, und wirft sie dann in eine exponentiell wachsende Belegschaft, um die exponentiell wachsende Nachfrage nach immer mehr Software zu befriedigen. Jeden Tag dringt das Kartenhaus, das wir Software nennen, tiefer und tiefer in unsere Infrastruktur, unsere Institutionen, unsere Regierungen und unser Leben ein. Und mit jedem Tag wächst das Risiko einer Katastrophe.

Von welcher Katastrophe spreche ich? Es ist weder der Zusammenbruch unserer Zivilisation noch der plötzliche Zerfall aller Softwaresysteme auf einmal. Das Kartenhaus, das einstürzen wird, besteht nicht aus den Softwaresystemen selbst. Vielmehr ist es das fragile Fundament des öffentlichen Vertrauens, das in Gefahr ist.

Zu viele weitere Vorfälle wie die 737 Max, Toyotas unbeabsichtigte Beschleunigung, Volkswagens Dieselaffäre oder der Iowa Caucus – zu viele weitere Fälle von aufsehenerregenden Softwarefehlern oder Fehlverhalten – und der Mangel an Praktiken, Ethik und Standards wird in den Fokus einer misstrauischen und wütenden Öffentlichkeit geraten. Und dann werden Vorschriften folgen: Vorschriften, die niemand von uns wollen sollte; Vorschriften, die unsere Fähigkeit, das Handwerk der Softwareentwicklung frei zu erkunden und zu erweitern, lähmen werden; Vorschriften, die das Wachstum unserer Technologie und Wirtschaft stark einschränken werden.

Es ist nicht das Ziel dieses Buchs, die überstürzte Einführung von immer mehr Software zu stoppen. Es ist auch nicht das Ziel, das Tempo der Softwareproduktion zu verlangsamen. Solche Ziele sind nicht der Mühe wert. Unsere Gesellschaft braucht Software, und sie wird sie bekommen, egal wie. Der Versuch, diesen Bedarf zu drosseln, wird die sich abzeichnende Katastrophe in Bezug auf das öffentliche Vertrauen nicht aufhalten.

Vielmehr ist es das Ziel dieses Buchs, Softwareentwicklern und ihren Managern die Notwendigkeit von Praktiken aufzuzeigen und diesen Entwicklern und Managern die Praktiken, Standards und Ethik beizubringen, die am effektivsten sind, um ihre Fähigkeit zu maximieren, robuste, fehlertolerante und effektive Software zu produzieren. Nur wenn wir Programmierer unsere Arbeitsweise ändern, indem wir unsere Praktiken, Ethik und Standards verbessern, kann das Kartenhaus gestützt und vor dem Zusammenbruch bewahrt werden.

Der Aufbau des Buchs

Dieses Buch ist in drei Teile gegliedert, die drei Ebenen beschreiben: Praktiken, Standards und Ethik.

Praktiken sind die unterste Ebene. Dieser Teil des Buchs ist pragmatisch, technisch und reglementierend. Programmierer aller Couleur werden vom Lesen und Verstehen dieses Teils profitieren. Auf den Seiten dieses Teils finden sich mehrere Verweise auf Videos in englischer Sprache. Diese Videos zeigen den Rhythmus der testgetriebenen Entwicklung und der Refactoring-Disziplinen in Echtzeit. Die zugehörigen Texte im Buch versuchen ebenfalls, diesen Rhythmus einzufangen, aber nichts eignet sich so gut für diesen Zweck wie Videos.

Standards sind die mittlere Ebene. In diesem Teil werden die Erwartungen, die die Welt an unseren Beruf stellt, umrissen. Er eignet sich besonders für Manager, damit sie wissen, was sie von professionellen Programmierern erwarten können.

Ethik ist die oberste Ebene. In diesem Abschnitt wird der ethische Kontext vom Beruf des Programmierers beschrieben. Dies geschieht in Form eines Eids beziehungsweise einer Reihe von Versprechen. Er ist mit vielen historischen und philo-

sophischen Erörterungen gespickt und sollte von Programmierern und Managern gleichermaßen gelesen werden.

Ein Hinweis für Manager

Diese Seiten enthalten viele Informationen, die für Sie von Nutzen sein können. Sie enthalten auch eine Menge technischer Informationen, die sie wahrscheinlich nicht brauchen. Ich empfehle Ihnen, die Einleitung jedes Kapitels zu lesen und die Lektüre abzubrechen, wenn der Inhalt Ihnen zu technisch wird. Gehen Sie dann zum nächsten Kapitel, und beginnen Sie von vorn.

Lesen Sie unbedingt Teil II, »Die Standards«, und Teil III, »Die Ethik«. Lesen Sie auch unbedingt die Einführungen zu jeder der fünf Praktiken.

Videos zum Buch

Geben Sie den untenstehenden Code unter

`https://mitp.code-load.de/`

in das Textfeld ein und klicken Sie auf EINLÖSEN, um die Videos zum Buch herunterzuladen (nur in englischer Sprache verfügbar):

`c6xea22a7x`

Danksagungen

Vielen Dank an meine unerschrockenen Rezensenten: Damon Poole, Eric Crichlow, Heather Kanser, Tim Ottinger, Jeff Langr, und Stacia Viscardi. Sie haben mich vor so manchem Stolperstein bewahrt.

Mein Dank gilt auch Julie Phifer, Chris Zahn, Menka Mehta, Carol Lallier und all jenen bei Pearson, die so unermüdlich daran gearbeitet haben, dass diese Bücher so gut gelungen sind.

Wie immer möchte ich mich bei meiner kreativen und talentierten Illustratorin Jennifer Kohnke bedanken. Ihre Bilder bringen mich immer zum Lächeln.

Und natürlich danke ich meiner lieben Frau und meiner wunderbaren Familie.

Über den Autor

Robert C. Martin (Uncle Bob) schrieb seine erste Codezeile im Alter von 12 Jahren im Jahr 1964. Seit 1970 ist er als Programmierer tätig. Er ist Mitbegründer von `cleancoders.com`, einem Anbieter von Online-Videotrainings für Softwareentwickler, sowie Gründer von *Uncle Bob Consulting LLC*, einem Anbieter von Softwareberatung, Schulungen und Skill-Entwicklungsdienstleistungen für große Unternehmen weltweit. Außerdem hat er als Master Craftsman bei *8th Light, Inc.*, einem Softwareberatungsunternehmen in Chicago, gearbeitet.

Martin hat Dutzende von Artikeln in verschiedenen Fachzeitschriften veröffentlicht und tritt regelmäßig als Redner auf internationalen Konferenzen und Fachmessen auf. Er ist auch der Schöpfer einer hochgelobten Serie von Lehrvideos auf `cleancoders.com`.

Martin hat viele Bücher verfasst und herausgegeben, darunter die folgenden:

- Clean Agile: Die Essenz der agilen Softwareentwicklung (mitp, 2020)
- Clean Architecture: Das Praxis-Handbuch für professionelles Softwaredesign (mitp, 2018)
- Clean Coder: Verhaltensregeln für professionelle Programmierer (mitp, 2014)
- Clean Code: Refactoring, Patterns, Testen und Techniken für sauberen Code (mitp, 2009)

- UML for Java Programmers (Pearson, 2003)
- Agile Software Development (Prentice Hall International, 2013)
- Extreme Programming in Practice (Addison-Wesley, 2001)
- More C++ Gems (Cambridge University Press, 2000)
- Pattern Languages of Program Design 3 (Addison-Wesley, 1997)
- Designing Object Oriented C++ Aplications Using the Booch Method (Prentice Hall, 1995)

Als führende Persönlichkeit in der Softwareentwicklungsbranche war Martin drei Jahre lang Chefredakteur des *C++ Report* und erster Vorsitzender der *Agile Alliance*.